JOURNAL OF SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE
J. Softw. Maint: Res. Pract. **11**, 55–72 (1999)

Research

Design Metrics as an Aid to Software Maintenance: An Empirical Study

ELAINE H. FERNELEY*
Information Technology Institute, The University of Salford, Salford, Greater Manchester M5 4WT, UK

SUMMARY

This paper proposes a technique for identifying areas within system designs that may be prone to future corrective or preventative maintenance activity. From that, this paper presents a set of measures that can be applied at the design stage of software development to support the production of more maintainable software. The measures are founded on the theories of coupling and control flow and have been empirically validated on a medium-sized industrial project. The results have shown that the control flow and export coupling measures are significantly good predictors of the future error rate. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: corrective maintenance; preventative maintenance; software design; coupling measures; control flow measures; error rate

1. INTRODUCTION

Software maintenance activity is inevitable, either to enhance the software by altering functionality (perfective maintenance), to adapt the software to cope with changes in the environment (adaptive maintenance), to correct errors (corrective maintenance) or to update the software in anticipation of future problems (preventative maintenance) (Pressman, 1994). Indeed, surveys indicate and estimates suggest that on average as much as 70% of a project's software budget is devoted to maintenance activity over the life of the software (Port, 1988; Bennett, 1990). Therefore, it is becoming increasingly important to consider future software maintenance activity as software is designed and developed. Increasingly, empirical evidence has highlighted a relationship between the application of design measurement and both more accurate cost estimation and enhanced future software maintainability (Grady, 1994; Al-Janabi and Aspinwall, 1993; Goodman, 1993).

Note that the term '*measure*' rather than '*metric*' is used in this paper. Metrics are derived from direct measurement of '*simple*' attributes such as length, number of decisions or number of operators. Measures are functions of metrics which are derived from indirect measurement and

*Correspondence to: Dr. Elaine H. Ferneley, Information Technology Institute, The University of Salford, Salford, Greater Manchester M5 4WT, UK. Email: E.Ferneley@iti.salford.ac.uk

CCC 1040–550X/99/010055–18$17.50

Received 30 April 1998
Revised 13 November 1998

can be used to assess or predict more complex attributes such as '*complexity*' or '*maintainability*' (Fenton and Pfleeger, 1997).

Therefore, the aim of this research is: 'to develop measures to support the designer in discriminating between system architectures to minimize future corrective and preventative maintenance activity'. Perfective and adaptive maintenance are an inevitable, proactive, part of software evolution and are beyond the scope of this work. This research focuses on assessing the future maintainability of the developed software by highlighting features in software designs that may attract subsequent corrective or preventative maintenance activity. The seminal work of Glenford Myers identified module strength (or cohesion) and module coupling as key factors in decomposing a program into a set of modules, module interfaces and module relationships (Myers, 1975, 1976, 1978). The design methodology developed by Myers and others (Stevens, Myers and Constantine, 1974) defined a set of seven categories for module strength (coincidental, logical, classical, procedural, communicational, functional and informational) and six categories for module coupling (content, common, external, control, stamp and data). Recently, Bierman and Ott (1994) have produced more sophisticated intra-module cohesion measures based on data slices. Furthermore, Troy and Zweben have used Myers' coupling categories as a foundation for quantifying coupling at the design stage using information from design structure charts (Troy and Zweben, 1981).

This interest in quantification early in the life cycle is due to the system design phase being the stage at which the system's 'signature' is developed as the system's architecture is specified. This phase takes as input the requirements specification and produces as output both high-level and low-level abstract models of the proposed system. High-level (inter-module) models show the system architecture and represent entities such as processes, functions and types as nodes, the communication (or coupling) between such entities is shown as directed links. Low-level (intra-module) models show the algorithmic design and use nodes to represent processing activity; the flow of control between such activities is represented as directed links.

In order to predict corrective and preventative maintainability using design measures based on inter-module and intra-module design features we must understand the relationship between a component's design characteristics and its maintenance behaviour. Intra-module complexity may be assessed by examining the depth of nesting and control flow structure. Inter-module complexity may be assessed by examining the degree of coupling. Therefore, the following measures are proposed: a measure to assess depth of nesting, a measure to assess control flow and a measure to assess the degree of coupling.

In summary, this paper presents a set of measures, founded on coupling and control flow theory, that can be applied at the design stage of software development. The set of measures are known as AIMS (automated information-flow measurement set). The underlying hypothesis is:

> *Measures based on coupling and control flow can be useful in identifying areas of system design that may be subject to future corrective or preventative software maintenance activity.*

The developed measurement set, AIMS, has been validated against quantitative commercial data. The data were from an academic publishing company and were part of their order processing system (121 modules consisting of between 4 and 685 lines of code). The independent variable against which AIMS was validated was the error rate. By using error-rate data, AIMS has, strictly speaking,

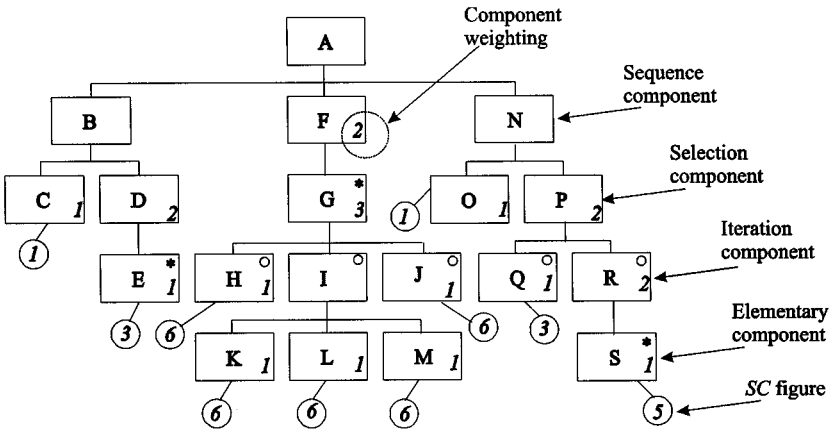


Figure 1. Jackson diagram illustrating the assignment of control flow measures

only been empirically validated against corrective maintenance activity. However, it will be argued that the work can also be extrapolated into the field of preventative maintenance. To support the application of AIMS to a wide range of system design techniques, a reconfigurable meta-CASE tool has also been developed, known as GIZMO.

2. DEPTH OF NESTING AND CONTROL FLOW MEASURES

2.1. Three intra-module measures

This research defined three intra-module measures:

- a '*raw*' depth of nesting measure,
- a refined depth of nesting measure that considered the specific combination of logical constructs that a unit of processing activity is embedded within, and
- a composite control flow measure that provides a single control flow complexity figure for a specific module.

While this research used Jackson structures for representing intra-module designs (see Figure 1), the underlying principles are generic (Jackson, 1975). Table 1 summarises the proposed intra-module measures.

2.2. Application of weights

2.2.1. Raw depth of nesting measure

A specific subset of processing activity is collectively referred to as an elementary component i. An example of an elementary component is '*S*' in Figure 1. The nesting depth of any specific

Table 1. AIMS intra-module definitions

Acronym	Explanation and definition
$DEC(i)$	Depth of elementary component i
$SC(i)$	Spinal complexity (control flow history) of elementary component i
$CF(i)$	$\sum_0^n SC(i)$ this provides an aggregated figure for the control flow complexity of the module i

elementary component ($DEC(i)$) was calculated by counting the number of logical constructs within which the elementary component is embedded. In the case of elementary component '*S*', $DEC(S)$ has a value of 2. Note that the specific set of processing activity that is collectively referred to as '*S*' is embedded within an iteration which in turn is embedded within a selection.

2.2.2. *Refined depth of nesting measure*

The '*raw*' depth of nesting count does not differentiate between the various types of logical constructs, as, for example, 2-way versus *n*-way selections. Therefore, a refinement of the raw depth of nesting measure that does consider the specific combination of logical constructs that an elementary component is embedded within was defined as the '*spinal complexity*' measure, $SC(i)$. The $SC(i)$ measure is derived by application of the following rules:

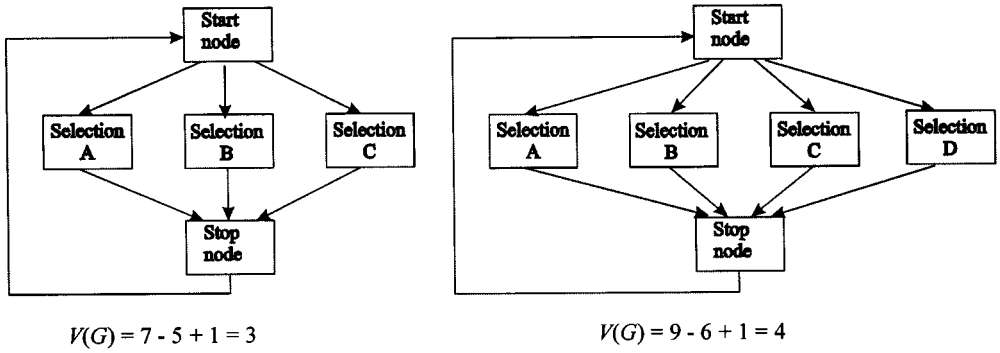
(i) The atomic elementary component is the lowest level representation of processing activity and therefore has an implicit complexity. This complexity is given a weighting of 1. As is illustrated in Figure 1, it is possible to represent such processing activity as either a single elementary component (e.g., '*C*') or as a series of sequential elementary components (e.g., '*K*', '*L*' and '*M*'). Such modelling is purely at the discretion of the designer. However, it can be assumed that if the designer feels the need to represent sequential processing activity as a series of elementary components then the implication is that there is some need to consider such processing as a number of distinct activities. Therefore, by weighting sequential elementaries distinctly any such subdivision by the designer would be considered in the weighting mechanism.

(ii) Sequential components within the body of a hierarchy diagram are not rated (for example, component '*B*' in Figure 1). They are a Jackson-specific feature which is a refinement of the parent, included for clarity in the design representation, and can be concatenated.

(iii) The mechanism for weighting logical constructs is derived from the cyclomatic complexity equation (McCabe, 1976) that may be applied to a strongly connected control flow graph (Berge, 1973). A strongly connected control flow graph is a graph where for any given pair of nodes (x, y) there is a path from x to y and a path from y to x. The cyclomatic complexity equation (v) for a control flow graph (G) is:

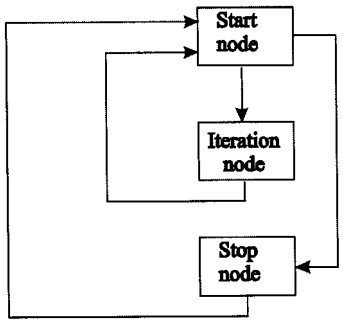
$$v(G) = e - n + 1 \tag{1}$$

where e is the number of edges in the control flow graph, and n is the number of nodes



$V(G) = 7 - 5 + 1 = 3$ $V(G) = 9 - 6 + 1 = 4$

Figure 2. Examples of selection weighting



$V(G) = 4 - 3 + 1 = 2$

Figure 3. Example of iteration weighting

in the control flow graph. By applying the cyclomatic complexity equation to strongly connected control flow graphs consisting of selection constructs, it can be seen that the resultant cyclomatic complexity figure corresponds to the number of selectable alternatives. For example, the illustrations in Figure 2 show strongly connected control flow graphs consisting of three and four selectable alternatives. For them, the associated cyclomatic complexity figures are also 3 and 4, respectively. Therefore, the weightings for selectable component nodes (or parent nodes) are determined dependent on the number of selectable alternatives emanating from the parent node. The larger the span of the selection, the higher is the weighting.

(iv) Similarly, application of the cyclomatic complexity equation to the strongly connected graph which represents iterative constructs results in a cyclomatic complexity value of 2 because either the iteration is or is not entered (Figure 3). Therefore, the weighting for iterative components is 2.

(v) Having assigned weights to each component within the intra-module system design, composite weights can be calculated for each path (or '*spine*'), thus giving consideration to

both the depth of nesting and the particular logical constructs that the elementary components are embedded within: $SC(i)$. For example, in Figure 1 the weight, $SC(K)$, for the path to elementary component 'K' is 6. This is calculated by adding the weights from all the components in the path: 'F' 2, 'G' 3, 'I' 0, and 'K' 1.

2.2.3. Control flow structure measure

Having defined a measure that considers the logical constructs and depth of nesting for each individual elementary component within a control flow structure, the research then defined a composite weight for the entire design: $CF(i)$. The $CF(i)$ value is calculated by combining the weights for each spine ($\sum SC(i)$), hence both depth of nesting, specific logical constructs and scope of logical constructs are considered (Ferneley, 1997). By using such an approach, control flow constructs that are '*higher*' in the intra-module system design (e.g., 'P' in Figure 1) have more influence on the final $CF(i)$ figure compared with those control flow constructs that are '*lower*' in the intra-module system design (e.g., 'R'). Therefore, the issue of intra-module '*scope*' is considered. In the example shown in Figure 1, the $CF(i)$ value is 43. This weighting mechanism allows intra-module structures to be assessed on an ordinal scale. The relation '$>$' can be applied to any given pair of intra-module diagrams.

3. COUPLING MEASURES

3.1. Three inter-module measures

This research defined a set of three high-level system design measures:

- a '*raw*' coupling measure which is simply a coupling count,
- a refined coupling measure which considers the structure of the data being passed, and
- a composite coupling measure which differentiates between a single input (or output) source and multiple input (or output) sources.

High-level system designs were chosen for assessment as they show the basic system entities and the communication between them. They are usually the first system architecture models produced. As such, researchers have focused on developing guidelines, rules and measures to support their development. Examples are the work by Myers (1975, 1976, 1978), Henry and Kafura (1981), Card and Agresti (1988), and Ince and Shepperd (1989). Henry and Kafura's work on import and export coupling has been used as a foundation for the coupling measures presented here (Henry and Kafura, 1981).

The theory is that module coupling occurs by data flow. Examples are parameter passing (local flows), the sharing of data structures (global flows), and that in an ideal system architecture such communication should be minimized to reduce future understanding and maintenance problems. Unlike Henry and Kafura's work, this research does not differentiate between local and global flows. Instead, this is based on the work of Lohse and Zweben who showed that there was no significant difference between the sharing of global data structures and parameter passing with reference to modifiability (Lohse and Zweben, 1984).

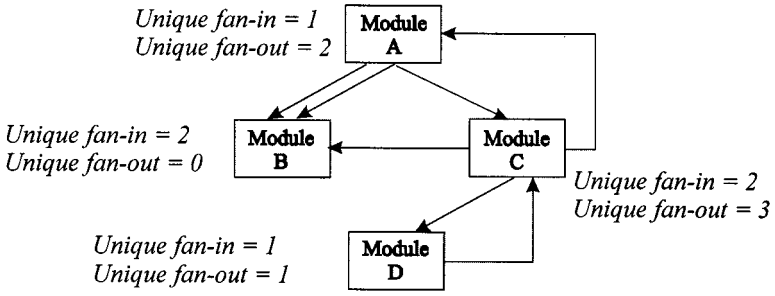


Figure 4. Example illustrating coupling

3.2. Application of weights

3.2.1. Raw fan-in and fan-out measures

This research uses the terms '*fan-in*' and '*fan-out*' to refer to import and export coupling, respectively. Fan-in is defined as: 'the number of unique local flows that terminate at a module M, plus the number of unique data structures from which data are received by M'. Fan-out is defined as: 'the number of unique local flows that emanate from a module M, plus the number of unique data structures that are updated by M'. Figure 4 diagrams fan-in and fan-out counting. The term '*unique*' is included in the definitions to avoid penalizing reuse (Ince and Shepperd, 1989).

It is postulated that as the number of fan-ins or fan-outs increases, so does the difficulty of understanding a module in isolation. For understanding the referenced global data structures and passed parameters, the '*ripple effect*' must be considered. However, should a global data structure or parameter be referenced more than once then only the initial call should be considered, otherwise reuse is penalized (Ince and Shepperd, 1989). Therefore, '*raw*' fan-in and fan-out measures can be defined as simply a count of the number of unique fan-ins FI to or fan-outs FO from the specified module, $M(i)$.

3.2.2. Weighted fan-in and fan-out measures

The implicit structures of fan-ins and fan-outs are not necessarily the same. Consider, for example, the difference between passing a single parameter and a complex variant record. Therefore, it is proposed that a refinement of the raw fan-in and fan-out measures should consider the structure of the data being passed. Such a weighting mechanism has been suggested by researchers including Myers (1975, 1976, 1978), Card and Agresti (1988), Shepperd and Ince (1993), and Hall and Preiser (1984). This research proposes that the fan-ins and fan-outs for module $M(i)$ are weighted by multiplying the FI and FO values by the $CF(i)$ value to incorporate the influence of the structure of the data being passed into or out of the specified module. The weighted individual fan-ins are summed to give an $FI\text{-}CF(i)$ value. An overall weighting for fan-outs ($FO\text{-}CF(i)$) is derived in the same way (see Table 2 for a summary of the definitions). By summing, relative ranking is maintained.

Table 2. AIMS inter-module definitions

Acronym	Explanation	Definition
FI-CF(*i*)	Fan-in complexity total of the module *i*	$\sum_1^N CF(i)$ complexity totals terminating at the module *i*
FO-CF(*i*)	Fan-out complexity total of the module *i*	$\sum_1^N CF(i)$ complexity totals emanating from the module *i*
FI-MCF(*i*)	Fan-in multiplicity complexity total of the module *i*	$FI\text{-}CF(i) * \sum_1^N fan\text{-}ins$
FO-MCF(*i*)	Fan-out multiplicity complexity total of the module *i*	$FO\text{-}CF(i) * \sum_1^N fan\text{-}outs$
CF(*i*)	Control flow complexity of each unique information flow terminating at (or emanating from) the module *i*	
N	Number of unique information flows terminating at (or emanating from) the module *i*	

Finally, it is proposed that consideration should be given to the difference between a single input source (or output destination) and a number of input sources (or output destinations). To achieve that, the *FI-CF*(*i*) figure is multiplied by the number of input sources to produce a fan-in multiplicity complexity total (*FI-MCF*(*i*)). A composite weighting for fan-out (*FO-MCF*(*i*)) can be derived in the same way (see Table 2 for a summary of the definitions). By using multiplication, the total number of unique paths into and out of the specified module are considered. Such composite measures provide a quantitative evaluation mechanism of import or export coupling and consider both the structure of the data being passed and the number of unique input sources or output destinations.

These weighting mechanisms allow import and export coupling features to be assessed on an ordinal scale. The relation '>' can be applied to any given pair of import or export coupling features.

4. SUPPORTING CASE TOOLS

4.1. GIZMO platforms

To support the application of AIMS a meta-CASE tool, GIZMO, was developed on a SUN MicroSystem ELC workstation with colour display. The graphical user interface was implemented in C within the X Window environment (OpenWindow version 3 with SUNOS UNIX version 4.13). The Xlib (MIT X Window System version 11, Release 4), Xt Intrinsics, Motif functions (OSF/Motif version 1.14) and standard Motif widgets were also extensively used. Prolog (BIM version 3.0) was used to create the validation rules and the diagram complexity evaluation mechanisms. The logical diagram structures for diagram validation and evaluation and the diagram repositories were

also implemented using Prolog predicates. A C parser implemented in Perl (version 4.0) was incorporated into the meta-CASE tool to allow design data to be reverse engineered from C source code.

4.2. Diagram analysers

Two interdependent analysers were developed, a control flow analyser and a data flow analyser, to acquire measurement data from Jackson diagrams. Figures 5 and 6 show the high level designs of the key elements of the two analysers. Pseudo-code was used for the designs; the designs were ultimately translated into Prolog.

Both syntactically valid and invalid Jackson diagrams can be stored, thus providing a '*sketch-pad*' type facility. A '*valid*' flag is associated with each type of diagram. When a diagram is created, the flag is set to false and remains so until the user validates the diagram. If the diagram is found to be syntactically correct then the flag is set to true and remains unchanged until the user amends the diagram again. Then the flag is automatically set back to false. Limited validation is required for data flow diagrams (for example, to check that links either terminate at and/or emanate from modules). Jackson diagrams used as diagrams of the control flow receive a more comprehensive validation to ensure that the rules of construction have been adhered to than do Jackson diagrams used as diagrams of the data flow. Only syntactically valid Jackson diagrams can be analysed.

Since the diagrams used for control flow are read from left to right, the position of each component is therefore important. The construction rules dictate that a component's '*children*' are all of the same type and that an iterated '*child*' should be unique in its instance. When the user activates the validation function, the name of the parent node is first recorded. The first child node is then checked and its type recorded. If the child type is iteration, then a check is made to ensure that it has no siblings. If there are siblings then the validation is halted and the error is indicated. If the child type is selection, then a check is made for more children. If there are no remaining children then the validation is halted and the error is indicated. This validation rule constrains the system developer to model an `ELSE` part of any `IF` (or `CASE`, etc.) statement even if the option is `ELSE` do nothing; such as the `ELSE` condition known as the `NULL` selection. Next, the types of the remaining children are checked. If a type differs from the recorded type of the first child then the validation is halted and the error is indicated. Finally, if the child type is sequence, a check is made for more children. If there are no remaining children then the validation is halted and the error is indicated. This validation rule enforces sequential decomposition. Additionally, component identifiers are checked to ensure that they are unique. The rules are recursively applied to each node, starting at the root node. Once a diagram has been validated its *CF* figure can be calculated. In order to calculate a specific node's weighting, its type and the number of children must be known, so that the weighting mechanism can then be applied. If a node has no children then it is defined as an elementary node and is automatically assigned a weighting of 1. Again, the rules are recursively applied to each node, starting at the root node. Having calculated the weightings of all the constituent nodes, the path figure ($SC(i)$) is passed back to the editor together with the name of the elementary node. Finally, the $SC(i)$ figures are accumulated to give the overall *CF* figure for the specific design.

Jackson diagrams used as diagrams of the data flow have a less comprehensive set of validation rules. At a syntactic level, any link within the diagram that neither terminates at or emanates from a node is invalid. Similarly, any node with no associated links is invalid. A link is implemented by


```
main:-
      check_diagram (diagram);
      evaluate (root, 0).

check_diagram (diagram):-
      get_nodes_in_diagram (diagram);
      find_root (nodes);
      validate (root).

validate (node):-
      get_children (node);
      apply_rule (node).

apply_rule (node, [0], [0], 0):-
      get_first_child_and_type (node);
      apply_child_rule (child);
      repeat
            get_next_node (child, [0], [0], 0);
            apply_child_rule (child);
      until no_more_children.

apply_child_rule (node, [0], [0], 0):-
      if type_iteration
            check_no_siblings (node, [0], [0], 0);
      if other_type
            check_for_siblings (node, [0], [0], 0);
            check_sibling_types (node);
      if error
            store (node);
            exit.

evaluate (node, complexity):-
      get_children (node);
      apply_cf_rules (node);
      add_node_complexity_to (complexity);
      if no_children
            store (node, 1);
      repeat
            evaluate (child, complexity);
      until no_more_children.
```

Figure 5. Pseudo-code for control flow analysis

including three '*invisible*' nodes on the link line, one at the beginning, one at the end and one in the middle (to provide a '*bend_node*' functionality). These three '*invisible*' nodes are provided to assist user selection of links. Although physically these three node types have similar characteristics to other nodes, as they do not contain any logical meaning (i.e., they are not really nodes), they are not stored inside the logical diagram structure. However, the '*invisible*' link nodes are still stored within the physical diagram structure, hence they support the diagram validation process. When the user activates the validation function all the link records in the physical diagram structure are checked (i.e., the physical diagram structure stores all the physical data about the current diagram). If there is any link where both the start and end '*invisible*' nodes have no corresponding position with actual nodes in the diagram, then the link may be identified as invalid. Additionally, component identifiers


```
main:-
       check_diagram (diagram;
       evaluate (node, [0], [0], 0, 0).

check_diagram (diagram):-
       get_nodes_in_diagram (diagram);
       find_first_node (nodes);
       validate (node).

validate (node):-
       get_links (link);
       apply_rule (link);
       repeat
              validate (link);
       until no_more_limks.

apply_rule (link):-
       get_invisible_nodes (link);
       repeat
              apply_invisible_node_rule (link);
              if error
                     store (link);
                     exit;
       until no_more_rules.

evaluate_fi_cf (node, [complexity], [0], 0, 0):-
       get_unique_fan_in_links (node);
       if no_fan_in_links
              store (node, [1], [0], 0, 0);
       apply_fi_cf_rules (node)
       add_fan_in_link_complexity_to ([complexity]);
       repeat
              evaluate_fi_cf (link, [complexity], [0], 0, 0);
       until no_more_fan_ins.

evaluate_fo_cf (node, [0], [complexity], 0, 0):-
       get_unique_fan_out_links (node);
       if no_fan_out_links
              store (node, [0], [1], 0, 0);
       apply_fo_cf_rules (node)
       add_fan_out_link_complexity_to ([complexity]);
       repeat
              evaluate_fo_cf (link, [0] [complexity], 0, 0);
       until no_more_fan_outs.

evaluate_fi_mcf (node, [0], [0], complexity, 0):-
       apply_fi_mcf_rules (node)
       add_fi_mcf_complexity_to (complexity).

evaluate_fo_mcf (node, [0], [0], 0, complexity):-
       apply_fo_mcf_rules (node)
       add_fo_mcf_complexity_to (complexity).
```

Figure 6. Pseudo-code for data flow analysis

are checked to ensure that they are unique within their type. Once a diagram has been validated, the range of data flow measures can be calculated. Individual import and export coupling features are weighted by invoking the control flow analyser to facilitate the application of the *CF* measure. As is illustrated in Figure 6, the individual figures are then stored in Prolog lists before being manipulated to provide composite import and export coupling weightings.

4.3. C parser

Access to realistically sized, industrial projects that were still at the design stage was limited, industry being notoriously reluctant to provide complete access to academics. The research as a whole looked at three commercial applications. The data presented here are from one of those applications: an order processing system developed by an academic publishing company (known as Project A). Ideally, empirical validation of AIMS would have been by the assessment of system designs prior to implementation and the subsequent monitoring of maintenance activity. However, Project A had already been implemented. The key advantage of using Project A was that a detailed corrective and perfective maintenance history had been kept. Therefore, design data were obtained by reverse engineering. Access to Project A's source code was restricted so the C parser was used to automate the collection of AIMS data without the need for detailed manual access to and analysis of the source code. From Project A's maintenance history, change density figures were kept for each module, where the change density, change rate or error rate (*ER*) is defined as: the total number of changes in a module, divided by the current number of lines of source code comprising the module.

For the purposes of this research, it is recognized that many changes in source code may be required as a result of a single fault. A fault occurs when a mistake in the software is identified as being a result of human error. An error may result in more than one fault, similarly a fault may result in more than one change being made to software. This situation is frequently referred to as the *ripple* effect (Kronlof, 1993). The terms *fault* and *error* have been used as specified in IEEE Standard 610.12 (IEEE, 1990). The term *current number of lines of code* was used as a normalising factor; obviously, as maintenance on a module is undertaken the number of lines of code will fluctuate.

Some parser-specific problems were encountered. When evaluating graphical representations of the design, compound conditions in system designs (for example: '`IF` *x* `=` *y* `AND` *a* `=` *b* `THEN`') were not recognized. In the example, a thinly disguised embedded `IF` statement was present. When implementing the parser, in order to maintain some consistency between the design and source code, Boolean operators were not considered. However, a minor change in the parser would allow for consideration of Boolean operators. Because the methodology was to apply AIMS to validated Jackson diagrams, the use of structured control flow was enforced. However, Project A's code included the occasional `break` and `goto` statement. Redevelopment of unstructured code was not possible so the use of unstructured control flow was allowed with each `break` or `goto` statement incrementing the *CF* measure by 1. Finally, using a design method, a set of sequential statements can be decomposed into a series of elementary components. Such a sequential decomposition is dependent solely on designer choice; this could not be accounted for in the parser implementation. Therefore, the parser recognized a sequential statement every time a semi-colon was reached.

5. THE RESULTS

5.1. Methodology

The set of measures, hereafter termed '*data set*', was ranked by the error rate and then compared with the various AIMS scores. The distribution of the error rate against the *CF* values is presented

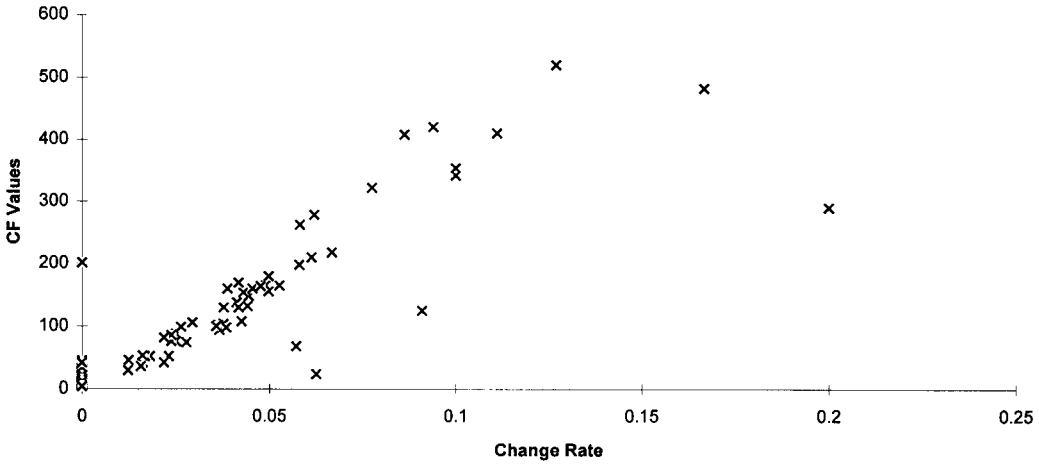


Figure 7. Scatter plot of control flow (CF) against measured error rate

in Figure 7. Spearman's rank correlation coefficient (r_s) and outlier analysis were used. Spearman's rank correlation coefficient was used because it is a non-parametric statistical technique using ranks rather than actual values, and the application of GIZMO resulted in data that were not normally distributed. In order to make the data conform to a normal distribution, and hence allow a wider range of statistical techniques to be applied to it, some practitioners propose the manipulation of the data using logarithmic or geometric transformations (Mosteller and Tukey, 1977; Myrvold, 1990). However, more recently researchers have suggested erring on the side of caution before applying logarithmic or geometric transformations (Shepperd, 1990; Fenton and Pfleeger, 1997). Indeed, Fenton and Pfleeger (1997) suggest that for '*measures of association*' where the distribution is non-normal either Spearman's rank correlation coefficient or Kendall's robust correlation coefficient should be used as they provide the most transparent means of analysing such data sets. Hence Spearman's rank correlation coefficient was used. The data items that were later identified as outliers were not removed from the data set before application of Spearman's rank correlation coefficient, such removal is inappropriate when applying ordinal measurement (Finkelstein, 1984).

Outlier analysis was chosen because '*anomalous*' results may provide interesting insights into maintenance problems. Outliers were identified by drawing boxplots (Fenton and Pfleeger, 1997). Boxplots were created by marking the median of the data set with a '+' and drawing a box from the lower to the upper quartile (the box represents the '*middle*' half of the data). An upper tail was defined by adding (1.5 times the box length) to the upper quartile. Similarly a lower tail was defined by subtracting the same amount from the lower quartile. The lower tail values were truncated where necessary to avoid meaningless values (for example, negative *CF* values). Values outside the upper and lower tails were defined as outliers. For an example, see Figure 8, that is, the box plot derived from the application of the *CF* measure to the Project A data. As shown in Figure 8, the key values for the generation of the boxplot were: median—61, lower quarter—8, upper quarter—130, lower tail—truncated to zero, upper tail—313. Seven values were identified as outliers.

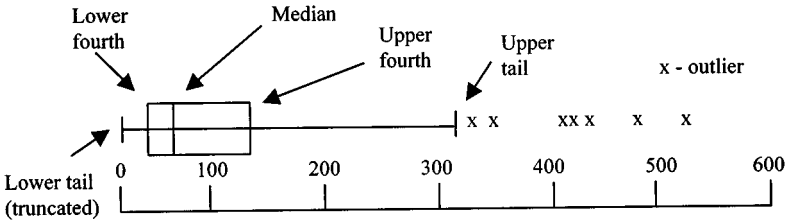


Figure 8. Boxplot of control flow (CF)

Table 3. Correlation of the error rates from staff ranking and measured ranking with AIMS values and the cyclomatic complexity ($v(G)$)

AIMS measures	Validation *ER* r_S		Significance level of difference of measured from				
	Staff	Measured	0.0000	*CF*	*FI*	*FO-CF*	$v(G)$
Mean *DEC*		0.5883	0.01	0.01	0.05	—	0.05
Mean *SC*		0.8235	0.01	0.01	0.01	—	0.01
CF	0.9034	0.8994	0.01	—	0.01	0.01	0.01
FI	0.4855	0.4458	0.01	0.01	—	0.01	0.01
FI-CF		0.4005	0.01	0.01	—	0.01	0.01
FI-MCF		0.4265	0.01	0.01	—	0.01	0.01
FO	0.5915	0.7116	0.01	0.01	0.01	0.01	—
FO-CF		0.8645	0.01	—	0.01	—	0.01
FO-MCF		0.8509	0.01	0.05	0.01	—	0.01
Non-AIMS $v(G)$		0.7137	0.01	0.01	0.01	0.01	—

5.2. Application of Spearman's rank correlation coefficient

The results of applying Spearman's rank correlation are presented in Table 3. The results of applying the various AIMS measures were correlated against the actual measured error rate ('measured') and a subjective *design quality* ranking provided by the developers ('staff'). The subjective ranking required the developers to assess both the original intra- and inter-module designs for their likely error rate. At the intra-module level, the developers were asked to rate the control flow *CF*. At the inter-module level the developers were simply asked to rate the fan-in *FI* and fan-out *FO* of each module; they were not asked to consider the structure of the data being passed or the number of input sources or output destinations. Obviously the developers were extremely familiar with the designs they were asked to evaluate as they had created them and were required to maintain them. However, module and component identifiers were removed from the design specifications used for the evaluation exercise in an attempt to encourage the developers to consider the designs in a more abstract way.

In the case of both the '*depth of elementary component*' (*DEC*) and the '*spinal complexity*' (*SC*) measures, the mean results were correlated against the measured error rate (*ER*) for each module. This was because the maintenance history data supplied restricted the analysis to the module level.

Specific locations of changes were not provided. The results show that there is some correlation between the error rate and depth of nesting with a '*raw*' depth of nesting (*DEC*) r_s value of 0.5883. Interestingly, the mean *SC* r_s value shows a significantly higher correlation result at 0.8235, suggesting that the specific logical constructs that an elementary component is embedded within have a bearing on the likely error rate.

The correlation coefficient results after assessing control flow structure against the error rate show significant correlations. Specifically, the *CF* r_s value for the data set is 0.8994. Whilst it is not surprising that there should be a correlation between control flow structure and error rate, it is reassuring to see such a correlation with the AIMS *CF* measure. Indeed, *CF* shows an improvement over the application of McCabe's cyclomatic complexity measure, $v(G)$ given in expression (1), where an r_s value of 0.7137 was derived on the same data set (McCabe, 1976). Therefore, with respect to this limited Project A data set, it can be concluded that the *CF* measure, which considers both control flow structure and the particular combination of logical constructs employed, is an excellent predictor of error rate.

The correlation coefficient results of assessing import coupling against error rate show weaker correlations (*FI*, *FI-CF* and *FI-MCF* r_s values of 0.4458, 0.4005 and 0.4265, respectively). These results concur with the findings of Card, Agresti and Glass who also found little relationship between error rate and fan-in (Card and Agresti, 1988; Card and Glass, 1990). Additionally, the subjective assessment of the fan-ins resulted in an r_s value of 0.4855, implying that developers (even when familiar with the modules being evaluated) find it difficult to equate fan-in structure to error rate. This result was particularly interesting because with the previous subjective evaluation of the intra-module designs, the developers had found it easy to recognize modules even though they had been '*disguised*'. This result suggests that developers do not concentrate on the fan-in of a module when they are developing it. When examining the relationship between export coupling and the error rate, reasonably strong degrees of association were found (*FO*, *FO-CF*, and *FO-MCF* r_s values of 0.7116, 0.8645 and 0.8509, respectively). Application of *CF* to the structure of the data being passed appears to significantly improve the correlation results although there is no significant difference between the *FO-CF* and *FO-MCF* figures. From these results, it can be concluded that whilst the structure of fan-outs is significant, the number of output destinations does not have a significant effect on future error rates. Interestingly, the subjective evaluation of the fan-outs resulted in an r_s value of 0.5915. As with fan-in, this suggests that developers can recognize the internal structure of a module more easily than its export interface and that developers find it difficult to predict the error rate from inter-module designs.

5.3. Outlier analysis

Outlier analysis using boxplots revealed a number of interesting results. Seven modules were identified with unusually high control flow values relative to low error rates. Since three of these modules were recent additions to the data set, the error rate was calculated over a shorter maintenance period. It had not been envisaged that new modules would have been introduced into the data set (they were introduced as a result of corrective maintenance activity). Subsequent discussion confirmed that these three modules and two others (both of which have error rates of zero) had been introduced to the data set since initial release. On the positive side this identification of the newly introduced modules as outliers confirmed the role of outlier analysis for the identification of

anomalies. A further three modules with low error rates relative to high *CF* figures were complex sorting routines. Investigation revealed that the algorithms used were mature and had not been developed in-house—hence, the low error rate. Additionally, one of the algorithms employed a large CASE statement which skewed the results; obviously the use of the CASE statement should not be penalized and future refinements of the measure may need to reconsider the allocation of weightings to large *n*-way selection statements. The final module identified as a *CF* outlier produces an exception report. Excellent implementation appeared to be the reason for its low error rate.

Seven modules were identified with unusually high fan-in values relative to low error rates. These outliers reinforce the argument that large fan-ins do not significantly contribute to the number of changes to which a module may be become subject. Finally, the modules with the two highest *FO-CF* and *FO-MCF* values also had the highest error rates. Furthermore, the developers ranked these modules in the bottom five in a subjective assessment of design quality, and regarded them as *maintenance time bombs*, and as a priority for future preventative maintenance activity.

6. CONCLUSIONS

This research builds on the seminal work of Myers and his associates (Myers, 1975, 1976, 1978; Stevens, Myers and Constantine, 1974) and is concerned with identifying relationships between inter-module and intra-module system designs and future maintenance activity. A set of three measures was defined that are based on coupling and control flow theory. Depth of nesting is measured by using a '*raw*' depth of nesting count and by refining the '*raw*' count to consider the specific logical constructs within which an elementary component is embedded. Control flow was measured by using a technique founded in linearly-independent path theory. Import and export coupling were measured separately by considering both the number of input sources or output destinations, and the implicit structure of the data being passed. The measures were validated by using data supplied by an academic publishing company, that had a total of 121 modules where maintenance history had been recorded. To aid in the application of the measurement set a CASE tool, GIZMO, was developed that facilitated the derivation of the measures from with system designs or directly from C source code.

The first observation is that, for the data set analysed, the hypothesis was broadly satisfied and that measures based on coupling and control flow can be useful in identifying areas of system designs that may be prone to future corrective maintenance activity (or changes). The refined depth of nesting measure showed an improvement over the '*raw*' depth of nesting measure when correlated against error rate (*SC*: r_s value $=$ 0.8235 against *DEC*: r_s value $=$ 0.5883). Similarly, the inter-module control flow based measure also showed an excellent correlation with error rate (*CF*: r_s value $=$ 0.8994). With respect to the developed coupling measures, the import coupling measures have no significant association with future error rates whilst the export coupling measures do. The low correlations between the various fan-in measures and the independent variable, error rate, confirm the findings of Card and Agresti (1988) who studied the relationship between coupling, fault rate and cost. Therefore, it is suggested that the number, scope and complexity of a module's import coupling has no bearing on its future maintainability. However, export coupling appears to have a much more significant affect on future maintainability. Specifically, the developed measures that consider the structure of the data being passed show an excellent correlation with error rate (*FO-CF*: r_s value $=$ 0.8645; *FO-MCF*: r_s value $=$ 0.8509).

Additionally, the hypothesis also considered preventative maintenance activity. By applying the developed fan-out and control flow measures to system designs it is proposed that preventative maintenance activity could be concentrated on those modules having high values for the measures.

To conclude, this paper has presented a set of method-independent measures founded on linearly-independent path theory that can be used for assessing both the implicit and explicit coupling and control flow of network and hierarchy models of system designs. The measures have been empirically validated against the maintenance history data of a commercial order processing system. The fan-out and control flow measures have been shown to be good predictors of the subsequent error rate.

This paper also provides a foundation for future research. Two areas are of specific interest. Firstly, the use of the developed measures for directing preventative maintenance activity. This would involve the identification of '*thresholds*' for the *CF* measure and the various fan-out measures. Should a measurement result fall outside the predefined '*threshold*' then preventative maintenance activity could be targeted towards the specific anomalous part of the system design. Secondly, further research into the location of errors relative to the specific combination of logical constructs that an elementary component is embedded within is an obvious future direction.

Acknowledgements

The author acknowledges the support of the Current Science Group and thanks them for allowing access to their software maintenance data. She also thanks the *Journal*'s referees for their instructive comments and additional insights.

References

Al-Janabi, A. and Aspinwall, E. (1993) 'An evaluation of software design using the DEMETER tool', *Software Engineering Journal*, **8**(6), 319–324.

Bennett, K. H. (1990) 'An introduction to software maintenance', *Information and Software Technology*, **32**(4), 257–264.

Berge, C. (1973) *Graphs and Hypergraphs*, North-Holland, Amsterdam, 528 pp.

Bierman, J. M. and Ott, L. M. (1994) 'Measuring functional cohesion', *IEEE Transactions on Software Engineering*, **20**(8), 644–657.

Card, D. N. and Agresti W. W. (1988) 'Measuring software design complexity', *Journal of Systems and Software*, **8**(3), 185–197.

Card, D. N. and Glass R. L. (1990) *Measuring Software Design Quality*, Prentice-Hall, Inc., Englewood Cliffs NJ, 129 pp.

Fenton N. E. and Pfleeger S. L. (1997) *Software Measurement: A Rigorous Approach*, 2nd edn., International Thompson Computer Press, London, UK, 638 pp.

Ferneley, E. H. (1997) 'An empirical study of coupling and control flow metrics', *Journal of Information and Software Technology*, **39**(13), 879–887.

Finkelstein, L. (1984) 'A review of the fundamental concepts of measurement', *Measurement*, **2**(1), 25–32.

Goodman, P. (1993) *Practical Implementation of Software Metrics*, McGraw-Hill International (UK), Ltd., Maidenhead, UK, 230 pp.

Grady, R. B. (1994) 'Hewlett-Packard—successfully applying software metrics', *IEEE Computer*, **27**(9), 18–25.

Hall, N. R. and Preiser, S. (1984) 'Combined network complexity measures', *IBM Journal of Research and Development*, **28**(1), 15–27.

Henry, S. and Kafura, D. (1981) 'Software structure metrics based on information flow', *IEEE Transactions on Software Engineering*, **SE-7**(5), 510–518.

IEEE (1990) *Glossary of Software Engineering Terminology, IEEE Standard 610.12*, IEEE Press, New York NY, 84 pp.
Ince, D. and Shepperd, M. (1989) 'An empirical and theoretical analysis of an information flow based design metric', in *Proceedings of the 2nd European Software Engineering Conference*, Springer-Verlag, Warwick, UK, 230 pp.
Jackson, M. A. (1975) *Principles of Program Design*, Academic Press, London, UK, 297 pp.
Kronlof, K. (1993) *Method Integration*, John Wiley & Sons, Ltd., Chichester, UK, 258 pp.
Lohse, J. B. and Zweben, S. H., (1984) 'Experimental evaluation of software design principles: an investigation into the effect of module coupling on system modifiability', *Journal of Systems and Software*, **4**(4), 301–308.
McCabe, T. J. (1976) 'A software complexity measure', *IEEE Transactions on Software Engineering*, **SE-2**(4), 308–320.
Mosteller, F. and Tukey, J. W. (1977) *Data Analysis and Regression: A Second Course*, Addison-Wesley, Manila, Philippines, 588 pp.
Myers, G. J. (1975) *Reliable Software through Composite Design*, Petrocelli/Charter, New York NY, 159 pp.
Myers, G. J. (1976) *Software Reliability: Principles and Practices*, John Wiley & Sons, Inc., New York NY, 360 pp.
Myers, G. J. (1978) *Composite/Structured Design*, Van Nostrand Reinhold Co., New York NY, 174 pp.
Myrvold, A. (1990) 'Data analysis for software metrics', *Journal of Systems and Software*, **12**(3), 271–275.
Port, O. (1988) 'The software trap—automate or else', *Business Week, Special Report*, **9**(1), 142–154.
Pressman, R. S. (1994) *Software Engineering: A Practitioner's Approach*, 3rd edn., European Adaptation, McGraw-Hill International (UK), Ltd., Maidenhead, UK, 801 pp.
Shepperd, M. (1990) 'Design metrics: an empirical analysis', *Software Engineering Journal*, **5**(1), 3–10.
Shepperd, M. J. and Ince, D. (1993) *Derivation and Validation of Software Metrics*, Clarendon Press, Oxford, UK, 167 pp.
Stevens, W. P. Myers, G. J. and Constantine, L. L. (1974) 'Structured design', *IBM Systems Journal*, **13**(2), 115–139.
Troy, D. A. and Zweben, S. H. (1981) 'Measuring the quality of structured design', *Journal of Systems and Software*, **2**(2), 113–120.

Author's biography:

Elaine Ferneley has recently become a Lecturer in Information Systems at the Information Technology Institute at the University of Salford, UK. She worked as a researcher in the Computation Department at UMIST from 1991 to 1994 and then joined the Department of Computing and Mathematics at Manchester Metropolitan University as a Lecturer. She had previously worked in industry in various systems analyst roles. Her present research interests are software measurement and the design of software agent technology. Elaine received her Ph.D. degree in Computation in 1996 and her Masters degree in Computation in 1991, both from the University of Manchester Institute of Science and Technology (UMIST). Her email address is: E.Ferneley@iti.salford.ac.uk

